Quiz Game App

Overview

An interactive quiz application with multiple categories, timed questions, scoring system, and performance analytics using in-memory questions and results.

Key Evaluation Criteria

- Correct provider type selection
- Proper state lifecycle management
- In-memory state persistence patterns
- Performance optimization (AutoDispose usage)
- State derivation and computation
- Code maintainability and organization
- Testing capability

Target Riverpod Concepts

- StateNotifierProvider for game state
- StreamProvider for question timer
- Complex score calculations
- Game state machines
- Provider.family for category-specific data

Functional Requirements

FR-01: Quiz Categories

- Browse 6-8 quiz categories (Science, History, Geography, Movies, Sports, Technology)
- View category details (question count, difficulty, high score)
- Select category to start quiz
- View completed categories with scores

FR-02: Quiz Session

- 10 questions per quiz session
- Multiple choice (4 options per question)
- Timer per question (15 seconds default)
- Skip question option
- Review mode (see correct answers after submission)
- Immediate feedback on answer selection

FR-03: Scoring System

- Points based on: correctness, time remaining, difficulty
- Bonus points for streaks (consecutive correct answers)
- Penalty for wrong answers (optional mode)
- Score multipliers for hard questions
- Final score calculation with breakdown

FR-04: Statistics and History

- Quiz history (date, category, score, percentage)
- Best score per category
- Overall accuracy percentage
- Average completion time
- Strongest/weakest categories
- Achievement badges (Perfect Score, Speed Demon, etc.)

Technical Requirements

TR-01: Provider Architecture

// Required providers to implement:

- quizCategoriesProvider (Provider<List<QuizCategory>>) *// Mock data*
- questionsProvider (Provider.family<List<Question>, String>) *// By category*
- quizSessionProvider (StateNotifierProvider<QuizSession?>)
- currentQuestionProvider (Provider<Question?>) *// Computed*
- questionTimerProvider (StreamProvider.autoDispose<int>) *// Countdown*
- answerSelectionProvider (StateProvider<String?>) *// Selected option*
- scoreCalculatorProvider (Provider) *// Service for calculations*
- quizHistoryProvider (StateNotifierProvider<List<QuizResult>>)
- categoryStatsProvider (Provider.family<CategoryStats, String>)
- achievementsProvider (Provider<List<Achievement>>) *// Computed*
- leaderboardProvider (Provider<List<QuizResult>>) *// Top scores*

TR-02: Game State Machine

// States: Idle, QuestionShowing, AnswerSelected, ReviewMode, Completed
// Transitions:
// Idle → QuestionShowing (start quiz)
// QuestionShowing → AnswerSelected (select answer)
// AnswerSelected → QuestionShowing (next question) or ReviewMode
// ReviewMode → Completed (finish review)

TR-03: Scoring Algorithm

- Base points: 100 per correct answer
- Time bonus: (timeRemaining / totalTime) × 50

- Difficulty multiplier: Easy ×1, Medium ×1.5, Hard ×2
- Streak bonus: (consecutiveCorrect - 1) × 25
- Wrong answer penalty: -20 points (if enabled)

TR-04: Mock Data

- 100+ questions across all categories
- Mix of difficulties (Easy, Medium, Hard)
- All questions have 4 options and 1 correct answer
- Sample quiz history (20+ completed quizzes)

Success Metrics

- ☐ Timer counts down accurately
- ☐ Score calculation is precise
- ☐ State transitions work correctly
- ☐ Streak detection works properly
- ☐ Statistics compute accurately
- ☐ Game state handles interruption
- ☐ Provider disposal on exit

Evaluation Rubric (100 points)

- Game state machine (25 pts)
- Timer implementation (20 pts)
- Scoring algorithm (25 pts)
- Statistics accuracy (15 pts)
- State management (10 pts)
- Code organization (5 pts)

General Assessment Framework

Code Quality Standards (All Projects)

Provider Organization

- Proper file structure (/providers, /models, /services)
- Clear provider naming conventions
- Appropriate provider types selected
- Documentation for complex providers

Performance

- AutoDispose used where appropriate

- No unnecessary widget rebuilds
- Efficient data queries
- Proper use of select() and watch()

Error Handling

- Try-catch in async operations
- User-friendly error messages
- Retry mechanisms where appropriate
- Error state UI

Testing

- Unit tests for providers
- Widget tests for UI
- Integration tests for flows
- Mock providers for testing

Submission Requirements

Each project submission must include:

1. Complete source code with comments
2. README with setup instructions
3. Architecture diagram showing provider dependencies
4. Test coverage report (minimum 70%)
5. Known issues and limitations documentation
6. Video demo (2-3 minutes)